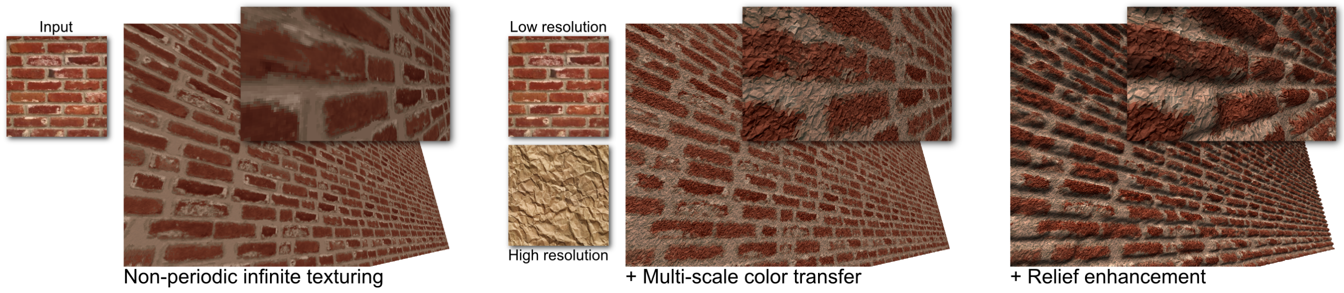
On-the-Fly Multi-Scale Infinite Texturing from Example

Kenneth Vanhoey, Basile Sauvage, Frédéric Larue, Jean-Michel Dischler*
ICube, Université de Strasbourg, CNRS, France



Abstract

In computer graphics, rendering visually detailed scenes is often achieved through texturing. We propose a method for on-the-fly non-periodic infinite texturing of surfaces based on a single image. Pattern repetition is avoided by defining patches within each texture whose content can be changed at runtime. In addition, we consistently manage multi-scale using one input image per represented scale. Undersampling artifacts are avoided by accounting for fine-scale features while colors are transferred between scales. Eventually, we allow for relief-enhanced rendering and provide a tool for intuitive creation of height maps. This is done using an ad-hoc local descriptor that measures feature self-similarity in order to propagate height values provided by the user for a few selected texels only.
Thanks to the patch-based system, manipulated data are compact and our texturing approach is easy to implement on GPU. The multi-scale extension is capable of rendering finely detailed textures in real-time.

CR Categories: I.3.3 [Computer Graphics]: Three-Dimensional Graphics and Realism—Viewing Algorithms I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture I.4.6 [Image processing and computer vision]: Segmentation—Region growing, partitioning

Keywords: Real-time Rendering, Infinite Texturing, Texture Tiling, Multi-scale Textures, Relief Textures, Color Space

Links: DL PDF WEB VIDEO

*e-mails: {Kenneth.Vanhoey,Sauvage,FLarue,Dischler}@unistra.fr

ACM Reference Format
Vanhoey, K., Sauvage, B., Larue, F., Dischler, J. 2013. On-the-Fly Multi-Scale Infinite Texturing from Example. ACM Trans. Graph. 32, 6, Article 208 (November 2013), 10 pages. DOI = 10.1145/2508363.2508383 http://doi.acm.org/10.1145/2508363.2508383.

1 Introduction

Providing efficient solutions for rendering detailed realistic environments in real-time applications, like games or flight/driving simulators, has always been a major focus in computer graphics. Details can be efficiently rendered using textures. But despite improvements of graphics hardware, memory capacity and data streaming techniques, which allowed over the recent years for increased scene complexity, texturing techniques must still fulfill constraints which are difficult to unify in a single approach. Ideally, they should 1) be as fast as possible to avoid penalizing frame rates, 2) use compact texture maps to limit streaming and data transfers that also penalize frame rates, 3) be non-periodic to avoid visual repetition artifacts, 4) produce fine details to avoid undersampling artifacts, 5) be enhanced with relief when details represent geometry like cracks or bumps to improve the rendering quality by accounting for parallax effects.

Meeting the five aforementioned conditions simultaneously is an open problem. Non-periodic real-time rendering of large surfaces, as well as compactness (properties 1, 2 and 3) can partially be solved by Wang tiling [Cohen et al. 2003] or corner tiling [Lagae and Dutré 2006]. However, to break the repetition effect one should provide tiles with multiple different borders or corners, and the number of tiles grows exponentially with that respect. Both compact and finely detailed textures (properties 2 and 4) can be defined thanks to multi-scale textures. However, stochastic tiling is not easy to extend to multi-scale texturing because border conditions must be made consistent throughout scales [Kopf et al. 2006]. Building the relief (property 5) is tedious without *a priori* knowledge if only a single image is available.

In this paper, we describe a new system to model textures that match all previous properties. It improves on stochastic texture tiling. Our system coherently includes the following contributions:

- Real-time non-periodic infinite surface texturing is achieved by modifying texture tiles at runtime, yet neither introducing visual artifacts nor requiring heavy computations. The memory consumption is also reduced compared to state-of-the-art tiling.
- Multi-scale texturing consistently blends colors between multiple layers of texture: our color transfer mechanism avoids popping and ghosting artifacts by respecting the features of each scale.
- Relief enhancement is integrated as a height map texture, which is kept consistent with the color texture.

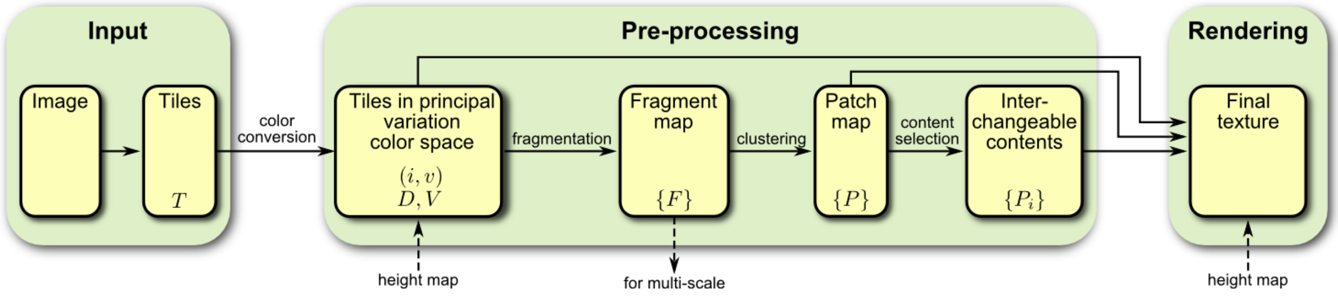


Figure 1: *Mono-scale infinite texturing flowchart and its interaction with the multi-scale extension and relief enhancement.*

In addition we provide a tool that makes it possible to interactively create a height map from a single color texture with a few clicks only. The user only needs to provide height values for a few selected points. These values are then propagated to the whole height map thanks to similarity between texture features. This tool is a useful complement to existing image-based modeling techniques.

In the remainder of the paper, after a review of related work (section 2), we give an overview of our system (section 3). We then describe the color space (section 4) used throughout the processing pipeline. Sections 5, 6 and 7 describe our main contributions, that is, respectively, our method for non-periodic infinite texturing, its extension to consistent multi-scale texturing, and relief enhancement together with our height map creation tool. Visual results, performance and limitations are discussed in section 8.

2 Related work

Texture synthesis and surface texturing are vast research areas. In the following, we briefly survey closely related works. We are concerned with image-guided, real-time, non-periodic and multi-scale surface texturing.

Non-periodic surface texturing. Two main streams can be distinguished: 1) parameter-based models used to directly define textures in the form of mathematical functions, 2) "by example" synthesis techniques.

The first category is well adapted to the parallel runtime computation of surface textures. Many models, *procedural* for most of them, have been developed from the mid-eighties on [Ebert et al. 2002]. Texture periodicity is avoided by introducing random functions, such as noise [Perlin 1985]. But some works [Bourque and Dudek 2004] argue that procedural textures are hard to define while automatic methods [Gilet and Dischler 2010; Galerne et al. 2012] are limited to a narrow range of patterns.

By example synthesis techniques [Wei et al. 2009] have been introduced to automate texture synthesis. The user supplies an example image to get a similar texture of larger size. These techniques require important computations, necessarily performed on explicit data arrays, because neighborhoods need to be accessible in order to account for local dependencies. This constraint makes runtime texturing much more intricate. Lefebvre and Hoppe [2005] introduce a fast parallel per-pixel technique. Non-periodic texturing of terrains is illustrated with additional clipmapping [Losasso and Hoppe 2004]. A multi-scale extension is introduced in [Han et al. 2008], allowing users to define the relationship between texture scales. Despite parallel implementation, computational requirements remain too high to allow for fast non-periodic infinite texturing of surfaces. To achieve real-time rendering, a drastic improvement in performance is needed. A common solution consists in separating the analysis and synthesis steps, as for Wang or corner tiling [Cohen et al. 2003; Lagae and Dutré 2006]. Tiles can be pre-computed and then randomly selected without propagating dependencies [Lagae and Dutré 2005]. It therefore outperforms classical pixel –or patch-based synthesis techniques. But with such tilings, repetition artifacts cannot be completely excluded if the number of different tile borders/corners is set too low (less than 4). This might often be the case for memory reasons since two (resp. three) different borders/corners require already 16 (resp. 81) tiles. The lack of variety in tile joints then results in a perception of the underlying rectilinear grid. Another issue is that recursive extensions, as in [Kopf et al. 2006], are not trivial when considering multiple arbitrary patterns representing different texture scales, because of complex dependencies among sub-tile contents in addition to dependencies on joints.

Smooth transition between multi-scale surface details. Providing real-time smooth transitions between multiple scales of surface details has been mainly addressed in the context of bi- or multi-scale material design [Wu et al. 2011]. It generally consists in making volumetric data (including bump/displacement textures) consistent with BRDFs. For color texturing (i.e. with no accurate relighting), as in our context, the usual solution trivially consists in using multi-resolution maps. Extremely large multi-resolution maps are common in the context of massive terrain data rendering [Treib et al. 2012] or mega-pixel texturing [Laine and Karras 2010], but they involve complex streaming mechanisms. Infinite runtime texture synthesis, that we are concerned with, avoids massive data management.

Image-guided textures. Image-guided approaches aim at allowing users to create textures from photographs [Bosch et al. 2011] using interactive 2D manipulations. But most methods ignore underlying relief, which is explicitly considered only for a small range of specific patterns [Müller et al. 2007]. For recovering accurate parallax effects during rendering, texturing techniques need an explicit relief representation [Chen et al. 2004]. To recover relief, specific photograph/scanner-based measurement devices have been developed [Ma et al. 2008], but they impose technical constraints. Extracting relief from a single image is more difficult, for instance with shape from shading (SfS) [Zhang et al. 1999]. It can be applied to texturing [Dischler et al. 2002]. But SfS makes strong assumptions concerning illumination, viewing conditions, surface smoothness as well as reflectance. The problem of SfS is known to be ill-posed [Zeng et al. 2005]. Painting relief by using interactive drawing tools is an interesting alternative solution for users. But these tools require painstaking manipulations to adequately choose brushes while texture features need to be drawn individually. [Brooks and Dodgson 2002] demonstrates that texture self-similarity can be used to facilitate color texture editing. We extend the use of feature self-similarity to facilitate relief texture editing.

3 Overview

In this section we give an overview of our infinite non-periodic texturing process, explain the coherence of its parts, and stress the usefulness of some tools. The process for mono-scale texturing is summarized in figure 1 as well as its interaction with the multi-scale extension and relief enhancement. The color conversion, fragmentation, clustering and content selection steps, which we summarize in this section, are pre-processed.

As an input we are given a set of tiles which are processed separately before the rendering stage. In this work we used a single periodic tile though other tilings (such as Wang/corner tiling) would also work in our process. The main idea is to randomly modify tile content on-the-fly at the rendering stage, so as to break the repetition effect. Therefore, we pre-compute a tile partitioning composed of *patches* for which we select a set of interchangeable *contents*. All contents for a given patch have the same shape, but correspond to different locations in the tile. They are chosen such that they can be exchanged while minimizing the appearance of seams. At rendering, each time a tile is repeated the content of its patches is randomly selected.

The multi-scale extension relies on this mono-scale process, which is first computed separately on several layers of texture, representing patterns viewed at different scales. For rendering, the colors of two consecutive layers are blended according to the viewing distance. It requires colors to be transferred between scales, usually from coarse to fine. Our contribution is a novel on-the-fly color transfer mechanism designed to minimize visual inconsistency (popping/ghosting and blocky artifacts) while blending. The key idea is that color is transferred not on individual texels but on pre-computed *fragments*, which are small pieces of texture, whose colors can be modified without introducing seams.

Fragments capture small features with a homogeneous color (*e.g.* a leaf or a petal in figures 4 and 8), which is decisive for multi-scale color transfer. On the contrary, patches must encompass several texture features (*e.g.* a few flowers) in order to break the repetition yet keeping visual feature arrangements. However, patches must be coherent with fragments, because content exchange (per patch) and color transfer (per fragment) will be combined at rendering. In addition they share a common objective: hiding seams, may they be due to content exchange or to color transfer. Here, we exploit the fact that seams are less perceivable in regions of high contrast: fragments are computed such that their borders match high contrasts. Patches are then defined as clusters of fragments.

A dedicated color space, that we call *principal variation color space*, is designed to meet several objectives: to facilitate the building of fragments, and the measure of contrasts and seams; to represent colors compactly; and to allow for improved multi-scale anti-aliasing. It separates a few dominant colors from local variations for each texture tile. It is used throughout the process: a tile is first and foremost converted into the principal variation color space, and colors are converted back only at rendering.

Relief enhancement is also integrated. If a height map is provided, it must be considered for the design of patches because content exchange may introduce geometric seams. Since patches are based on fragments which follow dominant colors, we introduce relief at the very beginning (when building the color space), as an additional coordinate to color channels.

4 Principal variation color space

As outlined before, we propose a new texture-dependent color representation. Similarly to SLIC superpixels' segmentation [Achanta et al. 2012] we extract a set of dominant colors by clustering, because it proved to be simple and efficient for extracting features. We combine it with an intra-cluster dimensionality reduction for capturing finer variations. More sophisticated clustering methods could be used instead. However they often put a lot of effort in avoiding over-segmentation, which is not a problem for fragmentation neither for exchanging content. In particular [Omer and Werman 2004] define clusters by lines together with a variation along it. We experienced however that texture color histograms do not exhibit such lines.

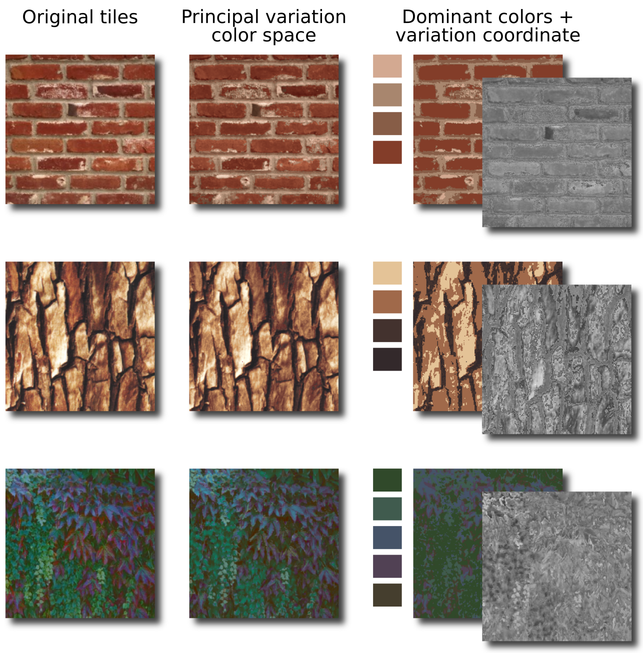

Original tiles
Principal variation color space
Dominant colors + variation coordinate

Figure 2: *Principal variation color space separates dominant color from local variations. Each color is replaced by an index i (referencing a dominant color and a principal variation), and variation coordinate v (right column). The original tile (left) suffers slight loss (middle) due to quantification and dimension reduction.*

Given an input texture in CIELUV space, we compute a table of color couples (D_i=dominant color, V_i=principal variation). A color C is then compactly represented by an index i and a variation coordinate v such that $C = D_i + vV_i$, as shown in figure 2. Dominant colors are intended to roughly classify features in the texture while local variations are described by v. It will be of peculiar importance for color transfer (section 6) that fragments respect these classes. Moreover blending dominant color and principle variation separately will improve anti-aliasing.

Given an input texture T, a clustering of texel colors is performed using K-means. Within each cluster i, a PCA produces a mean color D_i and three principal components: the principal variation V_i is the component associated to the largest eigen-value. The color $T(\mathbf{p})$ of any texel $\mathbf{p}$ is represented by its cluster index $i(\mathbf{p})$ and its coordinate $v(\mathbf{p})$ along $V_i(\mathbf{p})$. This approximation results in a squared error

$$\sum_i \sum_{\mathbf{p} \in \text{cluster } i} |T(\mathbf{p}) - (D_i + v(\mathbf{p})V_i)|^2 \tag{1}$$

which we would like to minimize. So we use $|T(\mathbf{p}) - (D_i + v(\mathbf{p})V_i)|^2$ instead of $|T(\mathbf{p}) - D_i|^2$ for clustering at each K-means iteration.

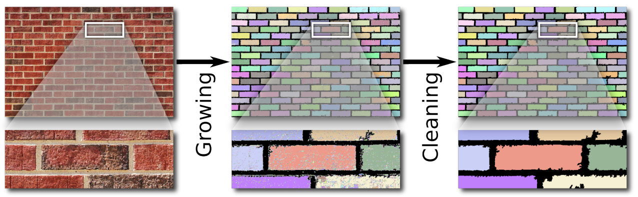


Figure 3: *Fragmentation proceeds in two steps. First, fragments are grown from seeds. Texels are added to a fragment when they do not exceed the contrast threshold value. Secondly, a cleaning step eliminates too small fragments by dispatching their texels in bigger neighboring fragments, so as to account for high-frequency noise.*

Defining the number of classes is left to the user: the number of dominant hues is a good guess and easy to determine. K-means is known to be sensitive to the initial values of D_i. We define a few initial sets by octree quantization [Gervautz and Purgathofer 1990] with different scalings of the luminance coordinate, and save the one that ends up with the lowest error. Results are shown in figure 2.

5 Non-periodic infinite texturing

Mono-scale infinite texturing (figure 1) processes input texture tiles independently. Each tile is partitioned into areas called patches, each of them having a set of interchangeable contents that can be randomly selected at rendering. The delicate task is to define the patches and their contents such that no seam is perceptible.

A tile is first converted into the principal variation color space using the procedure of section 4. A fragmentation stage (detailed in section 5.1) partitions the tile into small fragments which separate texture features. Each fragment has a uniform dominant color. Fragments are then clustered into patches of similar sizes, with irregular boundaries. At last, for each patch, a set of interchangeable contents are pre-selected in the tile in such way that an exchange will not generate new salient seams.

5.1 Fragmentation

Fragments are built by region growing. Figure 3 illustrates the two steps of fragmentation: region growing and cleanup of the smallest regions. The algorithm takes as an input a texture T and a contrast threshold τ.

Region growing. Regions F are grown around a seed. A texel $\mathbf{p}$ is added to F only if i) it is a 4-neighbor of F, ii) its color index $i(\mathbf{p})$ is the same as for the entire region, iii) its variation coordinate does not exceed the contrast threshold w.r.t. the average of the current region: $\|v(\mathbf{p}) - \frac{1}{|F|}\sum_{\mathbf{q}\in F} v(\mathbf{q})\| < \tau$, iv) the size of F does not exceed a maximum (see below). When no more texels of the border of the current region respect these constraints, the region is closed and now defines a fragment. The algorithm then iterates by growing a new region around a random texel among those bordering a previously created fragment, until all texels are accounted for.

Fragment cleaning. Some fragments can be too small, down to a few texels, due to noise and high frequency variations. A cleanup of these small fragments is therefore necessary. We proceed by emptying, as long as too small fragments exist, the smallest among them and dispatch their texels into the closest neighboring regions (in terms of average variation coordinate).

Parameter tuning. Since the computation time is low enough to provide interactive feedback, the τ parameter can easily be defined empirically. To do so, the user has to lower it until no fragments encompass multiple visual features that are significant to him (e.g. a brick, a petal, etc.). A tile will typically have a few hundreds of fragments in the end. The minimal size of a fragment can then be defined using a slider. The user increases it until the "noisy" fragments have vanished (see figure 3). Finally, in order to break too large fragments, the user lowers the maximal size (usually down to a several thousand texels).

The resulting fragment map has some useful properties. Firstly the fragments have similar sizes, which is important for subsequent clustering (section 5.2). Secondly all texels of a fragment have the same dominant color and the same principal variation (*i.e.* index). Thirdly the variation coordinate within a fragment is quite uniform. As a consequence fragments tend to capture features, and their boundaries often match high contrast in the texture. This results in texture masking that will greatly help in hiding seams, both for patch content exchange (section 5.2) and for multi-scale color transfer (section 6).

5.2 Patches with interchangeable contents

The core of our mono-scale texturing lies in the construction of a patch map together with interchangeable contents. Each patch P is a region of the texture. It is associated with a set of contents $\{P_i\}$ selected as regions of the same shape as P elsewhere in the texture. Therefore, the patches are chosen such that the content can be exchanged at rendering without introducing new visible seams. This is achieved by two means.
First the patch borders correspond to high contrasts, which are efficient in masking changes. An explanation for this is Weber-Fechner's law that states that the just-noticeable difference between two stimuli is proportional to the magnitude of these stimuli. For instance, assume we are exchanging the contents P_1 and P_2 in a patch P. The color change $P_1 - P_2$ may introduce a visible seam on the border ∂P. If the contrast around ∂P is high, then the perception of $P_1 - P_2$ is reduced.
Second, the contents are chosen such that dominant colors on the border coincide. We derive a measure of the visual impact that strongly penalizes the appearance of seams. That is, consecutive high changes along the border have an important impact on seam perception, conversely to randomly distributed high changes.

Note that our approach consists in a fixed patch shape for several contents, which is crucial for the upcoming real-time texturing. Indeed, a content can now be stored by a translation vector $\mathbf{t}$ only, meaning that $T(\mathbf{p})$ will be replaced at runtime by $T(\mathbf{p}+\mathbf{t})$ for all texels $\mathbf{p} \in P$. We can afford neither on-the-fly computation of the shape, nor storage of a shape per content. For this reason, methods that define an optimal cut from a content, such as graphcut textures [Kwatra et al. 2003] for instance, are not appropriate here.

Patch map. Patches are built from fragments by a straightforward clustering procedure. It is illustrated in figure 4 top: we define circles all over the tile using circle packing, and cluster fragments lying into them. As a consequence patch borders match fragment borders, which match high contrasts in the texture. A typical number of patches is 10 to 20 per tile. Some patches should encompass tile borders or corners: this is essential for breaking the repetition effect. For coherence at random tiling, the same patches must be present in all possible neighboring tiles, which is always trivially the case when using for example a single periodic tile.

Interchangeable contents. We want to select in T the best contents for P, *i.e.* the best translations $\mathbf{t}$. Therefore we need to measure the visual impact of the content replacement, *i.e.* the potential

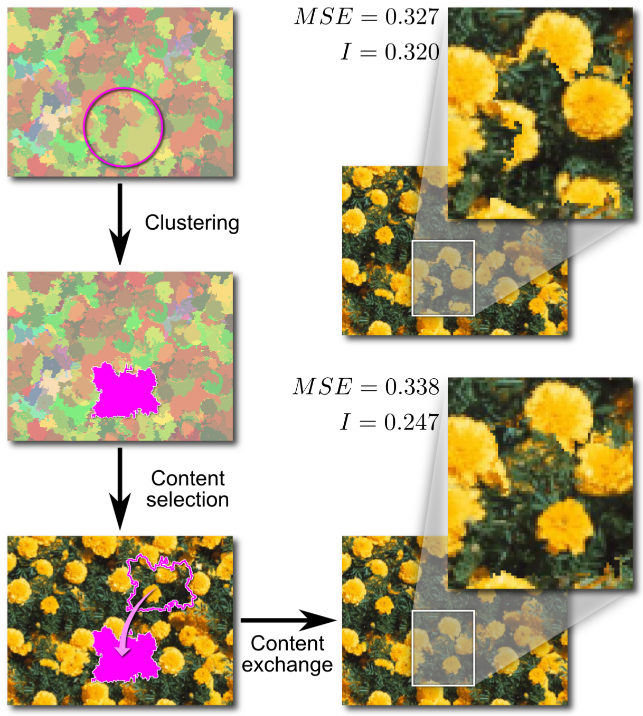


Figure 4: *Patches are built by fragment clustering and different contents are selected for exchange. Our measure I estimates the visual impact of content exchange more accurately than MSE, by detecting sequences of adjacent boundary texels with high errors.*

seams introduced at locations close to the border ∂P. We measure the impact by averaging an error over a 2-texel-wide dilation of the border:

$$I(\mathbf{t}) = \frac{1}{|dil(\partial P)|} \sum_{\mathbf{p} \in dil(\partial P)} E(\mathbf{p}, \mathbf{p}+\mathbf{t}) \quad (2)$$

The choice for E is discussed below. For each patch we compute N_t translations (actually 3 to 5) that minimize I. At runtime, the content of each patch is modified by randomly selecting one of these translations. Both memory consumption and computation time are thus kept low, while diversity is improved since a tile with N_P patches results in $N_t^{N_P}$ possible tile variants.

The per-texel error E must measure new seams while being quite robust to noise. A standard choice would be the color squared distance $\|T(\mathbf{p}) - T(\mathbf{p}+\mathbf{t})\|^2$ thus the average is the mean squared error. Its drawback resides in favoring a majority of low errors, regardless of the distribution of higher ones. The latter can however cause a more visible seam when packed together. Such a case is illustrated in figure 4. The squared distance between color gradients is also standard and has the same drawback.
We overcome this by using the classification defined for the dominant colors. We also compute a saliency map on the variation coordinate v using [Perazzi et al. 2012] from which we extract two saliency classes (by thresholding). We intersect the dominant color and saliency classifications to get a new set of N_C classes j. It is actually a refinement of the dominant color classes by a factor at most two. Let C_j be the average color of class j. We define

$$E(\mathbf{p}, \mathbf{p}+\mathbf{t}) = (M * \delta_{\mathbf{t}})(\mathbf{p}).\delta_{\mathbf{t}}(\mathbf{p}) \quad (3)$$

where $\delta_{\mathbf{t}}(\mathbf{p}) = \|C_{j(\mathbf{p})} - C_{j(\mathbf{p}+\mathbf{t})}\|^2$ is the map of class mean color errors, and where M is a magnification filter (we used a 7×7 mask with ones everywhere).
This error can be understood as follows. The use of class mean colors $C_{j(\mathbf{p})}$ instead of $T(\mathbf{p})$ serves as low-pass filter and quantization, such that the impact I is not disturbed anymore by majority of texels with small color changes. The refinement of the dominant colors' classification (using saliency) captures important features in v that are not present in dominant colors (*e.g.* see the dark region on the bricks in figure 2). The factor $M * \delta_{\mathbf{t}}$ magnifies the error $\delta_{\mathbf{t}}$ such that: non-seam texels $\delta_{\mathbf{t}}(\mathbf{p}) = 0$ are unchanged; isolated seam-texels $\delta_{\mathbf{t}}(\mathbf{p}) > 0$ are squared (because $M * \delta_{\mathbf{t}}$ equals $\delta_{\mathbf{t}}$ at $\mathbf{p}$); non-isolated seam-texels $\delta_{\mathbf{t}}(\mathbf{p}) > 0$ are squared and magnified (linearly w.r.t. the number of neighboring seam-texels, *i.e.* the length of the seam). As a consequence the visual impact I detects sequences of adjacent texels with high errors, which are typical visible seams (see figure 4).

5.3 Rendering

The GPU has at its disposal for each tile $i)$ the color texture; $ii)$ a patch map which is a texture containing, for each texel, the index of the patch it belongs to and a displacement vector towards the patch center; $iii)$ the table of contents (*i.e.* translations). At rendering the tiles are repeated according to the chosen tiling algorithm. Let $\mathbf{p}$ be a texel belonging to a patch of center $\mathbf{c}$, which lies in the k-th repetition of a tile. We need a random index in the table of contents: it must vary with $\mathbf{c}$ and k (avoid repetition) but not with $\mathbf{p}$ (the same content for the whole patch). This is done using a linear congruential pseudo-random number generator, initialized by a hashcode computed from $\mathbf{c}$ and k. Because patches might overlap tile borders, it is important that k is determined from the patch center $\mathbf{c}$ and not from $\mathbf{p}$. This way, every texel of the patch generates the same number, regardless of which tile repetition it is in.

6 Multi-scale texturing

In this section we show how the technique presented in the previous section can be used as a basis for multi-scale texturing. For the clarity of explanation we present our method for two scales. The main idea is illustrated in figure 5: the structure of the texture is taken into account when transferring colors.

Assume that two input images underwent the process of section 5. Thus at the rendering stage, two infinite textures T^H (high resolution for close-ups) and T^L (low resolution for far-away viewpoints) can be generated independently. In order to be able to blend the two textures without visual artifacts, one has to make sure that the color of a texel $\mathbf{q} \in T^L$ equals the average color of its footprint R on T^H:

$$T^L(\mathbf{q}) = \frac{1}{|R|} \sum_{\mathbf{p} \in R} T^H(\mathbf{p}) \quad (4)$$

This condition is easily met in cases where multi-resolution textures can be pre-synthesized and stored in very large data arrays. It mainly consists in shifting the average color over R onto $T^L(\mathbf{q})$. It is more computationally intensive to do this when textures are computed at runtime using stochastic tiling. Indeed, it would require to compute the right-hand side of equation (4) on-the-fly, which is not affordable. To avoid this we assume that the scale ratio is large enough for R to cover a large part of the tile T^H. So we can statistically approximate the average over R by the average $\overline{T^H}$ over the entire tile T^H. Consistent color transfer is processed at rendering by shifting the color of any fine texel $\mathbf{p}$:

$$T^H(\mathbf{p}) \leftarrow T^H(\mathbf{p}) + \left(T^L(\mathbf{q}) - \overline{T^H}\right) \quad (5)$$

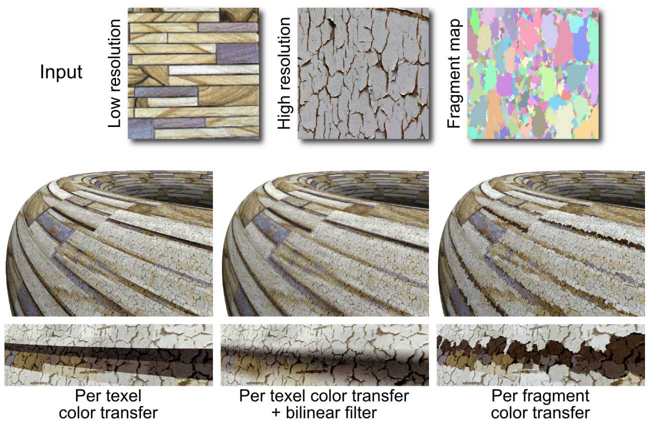


Figure 5: *Top (input): colors from a texture T^L (wood) are transferred to a texture T^H (cracks). Per-texel: modifying individual texel colors results in undersampling artifacts. Per-texel and bilinear filter of T^L: blocks are blurred but the features of T^H are not respected. Per-fragment: modifying fragment colors improves consistency, as if T^L was made of T^H.*

Doing this for all fine texels $\mathbf{p} \in R$ results in undersampling artifacts (limit between two coarse texels $\mathbf{q}$), as illustrated in figure 5 left. Blocks can be hidden by blurring T^L using a bilinear filter, as in figure 5 middle, but this produces ghosting. Bi-lateral filtering on T^L will not avoid these artifacts either: it operates de-noising on the low resolution, while the point here is smoothing and oversampling. Actually, any method that does not take the structure of T^H into account creates the impression that T^L is painted onto T^H, whereas we would like that T^H seems to constitute T^L. We therefore exploit the fragment map of T^H: the color of fragments can be modified without introducing artifacts because they have been computed accordingly. Thus we apply the same color shift for all texels of a fragment (figure 5 right), which hides the color changes on the fragment borders (no blocky artifacts) and increases the consistency between scales (no ghosting). As shown in figure 6, we do this by choosing $\mathbf{q} \in T^L$ in equation (5) corresponding to the center of fragment F containing $\mathbf{p} \in T^H$, not corresponding to $\mathbf{p}$ itself. In other words we relax the footprint of $\mathbf{q}$ to match the fragments borders.

At rendering, a smooth transition between T^L and T^H is computed by linear interpolation according to the viewing distance. To further reduce ghosting we exploit the color space: we fade the variational component out more quickly than the dominant color. The variational color space brings even more flexibility, such as detail enhancement (scaling v) or different tuning for every dominant color. All computations are done on the GPU by using only four texture accesses: one to T^L, one to T^H, one to the fragment map, and again one to T^L at the location of the center of the fragment in order to modify the color of T^H.

This principle can be recursively extended to an arbitrary number of scales. Figure 11 shows an example with 3 scales. The number of texture accesses grows linearly with the number of scales.

7 Relief enhancement

Assuming that height maps corresponding to the input textures are available, the system presented so far can be easily extended in order to integrate them properly.

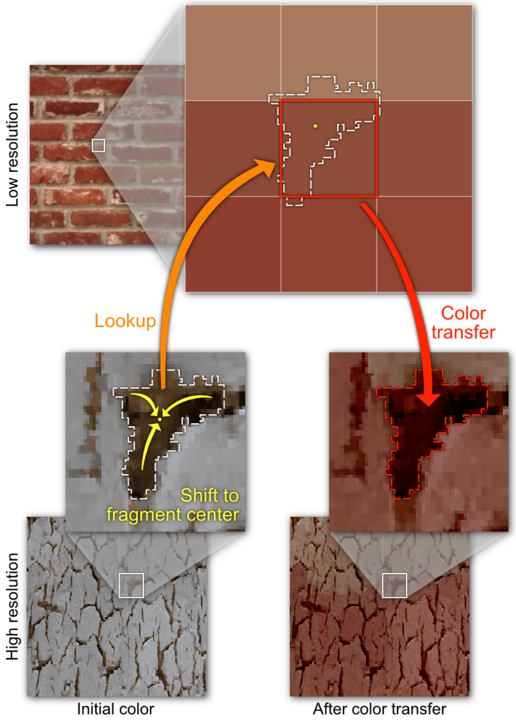
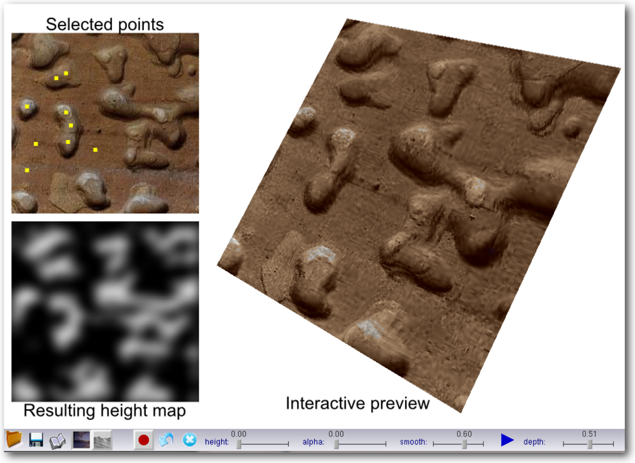


Figure 6: *Multi-scale texturing: color is transferred from low to high resolution. The same average color is transferred for the whole fragment: a first lookup at the fragment map (high resolution) leads to the fragment center; a second lookup at the low resolution texture defines the new average color.*

Pre-processing. If relief information is provided, salient features may be either chromatic or geometric. Hence, care has to be taken in order to account for both while creating fragments and patches. To do so, the height map is introduced as a fourth component (in addition to the three color channels) in the fast quantization during the color conversion stage. Since the color space is used for the following stages, both fragments and patches are coherent with geometric features.

Rendering. Our texturing method is compliant with existing relief rendering approaches. We used the relief mapping algorithm [Policarpo et al. 2005], which only requires minor modifications of the pixel shader. However, since this algorithm locally shifts texture coordinates according to the viewing direction in order to account for parallax effects, using height maps for other levels than the coarsest one is not trivial. Therefore we do not consider multi-scale for relief rendering: we first compute the texture coordinates shifting from the coarsest level height map, and then texturing is performed as described in the previous sections by using the modified coordinates.

Height map creation tool. In the case where no height map is available, we propose a simple approach to create one from a given texture. The principle is the following: i) a local descriptor is computed at every texel to characterize features, ii) the user provides height values for a few selected texels, iii) the provided values are propagated to the whole texture according to the similarity between local descriptors. The computation is fast enough to provide interactive edition (see figure 7) such that the user can refine the map by adding points iteratively.



height: 0.00 alpha: 0.00 smooth: 0.60 depth: 0.51

Figure 7: *Height map creation tool. A few points are manually selected by the user and assigned a height value. The full height map is then automatically computed by averaging specified height values according to our similarity measure. An interactive preview gives the possibility to iteratively refine the result by selecting additional points.*

This tool acts as a complement to usual tools like shape from shading (SfS), which are not always well suited: they perform poorly when color variations are barely related to illumination, they may also rely on expert knowledge for parameter tuning and often require lighting conditions to be estimated. Like SfS, we assume that textures are free of perspective or surface curvature related distortions, but illumination conditions can be arbitrary. We consider cases where relief is related to the image structures. Since textures generally show strong self-similarity [Brooks and Dodgson 2002], we assume that similar features are likely to have similar heights. This assumption holds for many cases of real-world textures: we perform well on the brick wall and bark textures for instance, while SfS fails to reconstruct a convincing height map. SfS would on the contrary work better on textures like the crumpled paper.

The similarity descriptor proposed for texture editing [Brooks and Dodgson 2002] is sensitive to noise, intensity, scale and rotation, which is too constraining for our purpose. Other texture descriptors, for instance based on filterbanks [Tou et al. 2009], aim at classifying patterns (i.e. feature arrangements). Conversely, we want to classify individual features similarly to what operators like SIFT do to characterize local image structures [Lowe 2004]. Thus we use local gradients to characterize texels. SIFT is however invariant to rotations and scales, which might be too loose for textures. Our local descriptor remains therefore sensitive to size and rotation, except for small angles or scale variations. It is computed from the variation coordinate v and is applied independently on each dominant color cluster (see section 4).

Considering image v corresponding to our variation coordinate, we build a pyramid of Gaussians by convolving v by kernels of increasing size σ: $v_\sigma = G_\sigma * v$. The descriptor of a given texel $\mathbf{p}$ is then expressed as the gradients of v_σ computed at $\mathbf{p}$ and its 8 neighbors:

$$\{\nabla v_\sigma(\mathbf{p}+\mathbf{d})\}_{\mathbf{d}\in\{-1,0,1\}^2,\ \sigma\in[1,N_\sigma]} \tag{6}$$

In practice, we use $N_\sigma = 6$. To evaluate similarity with another texel $\mathbf{p}'$, we compute a distance consisting of a sum of lengths of gradient differences:

$$d(\mathbf{p},\mathbf{p}') = \sum_{\sigma,\mathbf{d}} \left\|\nabla v_\sigma(\mathbf{p}+\mathbf{d}) - \nabla v_\sigma(\mathbf{p}'+\mathbf{d})\right\|. \tag{7}$$

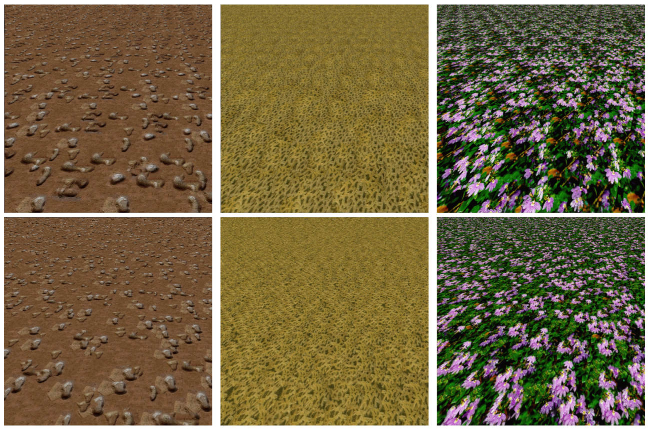


Figure 8: *Comparison to Wang tiling. Top: the underlying rectilinear grid remains perceivable with Wang tiles using 2 different borders (16 tiles). Bottom: our method (3 contents for 14, 10 and 18 patches respectively) discards repetition effects with only one tile.*

To allow for small rotations we compute d w.r.t. $\pm\pi/8$ rotated descriptors and keep the minimal value.

Based on a few selected texels for which height values are specified by the user, the final height map is reconstructed by propagating these values to the whole texture. In order to avoid staircase effects, the height of each texel is defined as the weighted average of the three closest selected texels (in terms of the distance d within its dominant color cluster), with weights inversely proportional to d.

8 Results

Mono-scale texturing. Figure 8 shows examples of infinite texturing (bottom) compared to Wang tiling (top). It shows that Wang tiles with only two borders cannot sufficiently break the visible grid artifact (the same applies to corner tiles with only 2 corners), whereas stochastic distribution solves well this problem in our case at a much lower memory cost. Seams are hidden by texture masking, as expected by our design of fragments, patches and content selection. As a result, patch content exchange generates consistent non-periodic textures even for those having complex realistic features (see flowers in figure 8 bottom).

Multi-scale texturing. The blending between two textures representing different scales is illustrated on several examples in figures 9 and 10. The coarse scale colors are propagated to the fine scale such that rendering is consistent across the resolutions. Fine details are consistently added in the close views: the transition between the scales is smooth and free of artifacts.

Though our method provides no high-quality pre-computed texture synthesis but on-the-fly generation and blending, our results are competitive w.r.t. Multiscale Texture Synthesis (MTS) [Han et al. 2008]. Figure 11 shows a comparison between both techniques using two and three input textures. Compared to MTS, our method has the advantage of texturing infinite surfaces on-the-fly. We can also interactively change all parameters: blending functions, scale ratios, color transfer functions, etc. Moreover, our color space allows for an increased flexibility in choosing how individual dominant colors are transferred and blended. Because of linearity, dominant colors and local variations can be handled separately and independently in formula 5. For instance, the user can choose a different color transfer, as illustrated in left-hand side example of

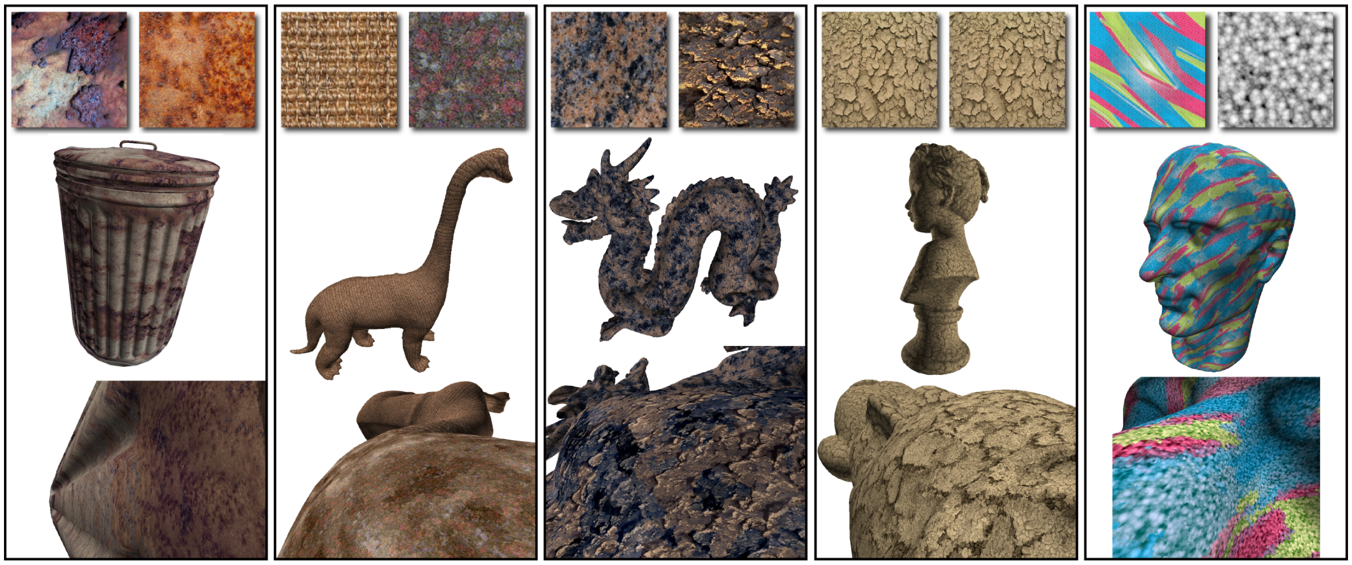


Figure 9: *Multi-scale texture blending: two textures (top) are smoothly blended and consistently rendered at different resolutions.*

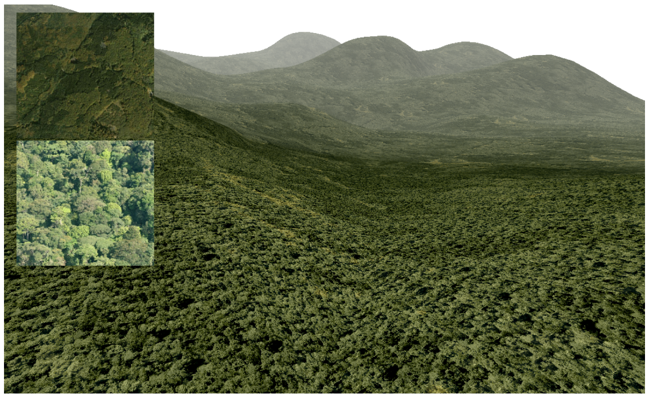


Figure 10: *Multi-scale texturing on an "infinite" surface.*

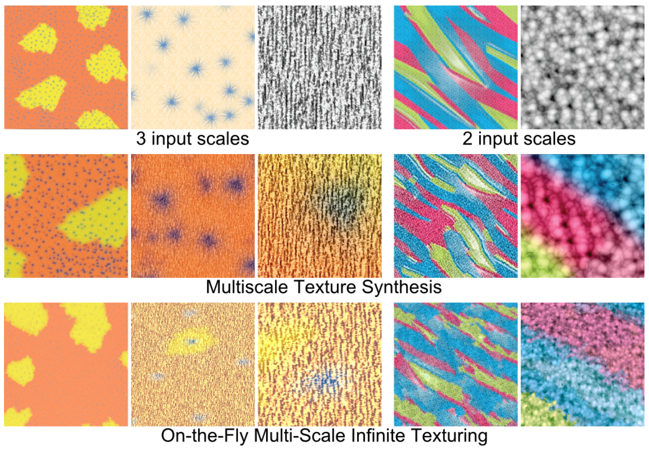

3 input scales
2 input scales
Multiscale Texture Synthesis
On-the-Fly Multi-Scale Infinite Texturing

Figure 11: *Comparison to "Multiscale Texture Synthesis". Though we only use blending, our results are visually competitive. The key is that our approach is an order faster and works on-the-fly.*

figure 11: we preserve the blue dominant color of the intermediate scale, while transferring yellow and orange from the coarse scale. The variational component may also be individually controlled so as to force variations' preservation from either scale.

Relief enhancement. Figure 12 shows how relief enhancement improves realism compared to flat texturing. The addition of geometric details is especially visible on silhouettes and shading effects. Using our interactive height map creation tool (figure 7), less than 10 selected texels and corresponding heights, assigned in less than 30 seconds were necessary to generate realistic height maps.

Performance. Any of the pre-processing stages is computed within a few seconds while rendering is real-time. To assess the impact of each of the contributions presented above, we measured rendering frame-rates for different texturing methods (from no texturing at all up to our multi-scale relief-enhanced texturing). Measurements were made for a 1200 × 800 framebuffer on a laptop with a GeForce GT555M GPU having 2GB of dedicated RAM, thus demonstrating that no latest high performance graphics board is needed to reach interactivity.

Model # triangles	Forest 12800	Caesar 33134	Dino 47960	#textures	memory	#accesses
No texturing	65 fps	31 fps	22 fps	0	0	0
Wang tiling (16 tiles)	88%	84%	86%	1	16	1
Our method (1 scale, 1 tile)	83%	84%	86%	2	2	2
Our method (2 scales)	46%	45%	45%	5	5	7
Our method (2 scales + relief)	25%	35%	41%	5	5	7+24

Table 1: *Comparison of performances (relative frame rates of the rendering stage). The number of textures, the memory load (number of tiles) and the number of texture accesses in the pixel shader are shown. Object names refer to figures 9 and 10.*

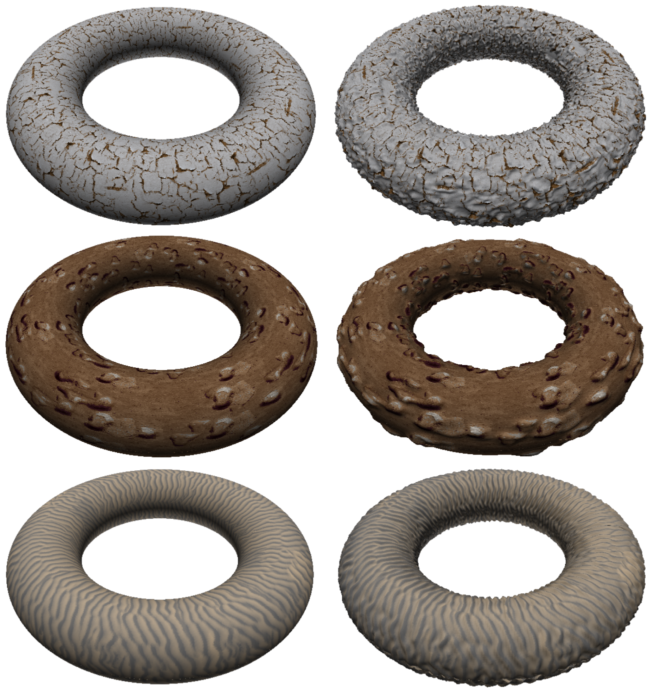


Figure 12: *Relief enhancement significantly improves realism. Left: flat texturing. Right: relief-enhanced texturing. Height maps have been reconstructed interactively in less than 30 seconds. From top to bottom, 8, 9 and 4 clicks were respectively necessary.*

Table 1 compares timings for three objects. Absolute frame-rates differ for each object depending on the number of pixels covered. Mono-scale patch content exchange is of equivalent speed compared to Wang tiling, though requiring a second texture access (for the patch map). Adding a second scale of resolution divides the frame-rate by two due to additional accesses. Each new scale requires three more texture maps (color texture, patch map and fragment map). Relief enhancement with parallax effects has been computed for the coarse texture only. It is done by (ray-tracer based) relief mapping [Policarpo et al. 2005] requiring many texture accesses. Implementing a geometry shader would drastically improve the performance.

Memory cost. Each input tile (one per scale) requires two GPU texture maps: one 3-channel texture map for storing the dominant color index, the variation and the height, and a 2-channel texture map for storing fragments. Each texture scale requires subsequently at most three tiles (to be compared with at least 16 tiles for Wang tiling). A two-scale rendering required at most 12MB of memory for the examples of this paper. This amount increases when additional mipmapping is used.

Limitations. Fragmentation is based on the assumption that there are sufficient contrasts in the texture image. If the image contains too smooth variations (low contrasts), fragmentation will fail, resulting in too few and too large fragments. The maximal size parameter (section 5.1) will then split them up and seams may become apparent at rendering. Figure 13 shows such an example.
A surface parameterization is also required by our approach, making it prone to texture distortions and discontinuities. Finally, relief enhancement only considers height fields. This geometric representation is limiting since it cannot represent complex structures such as fur, raffia weaves or yarn.

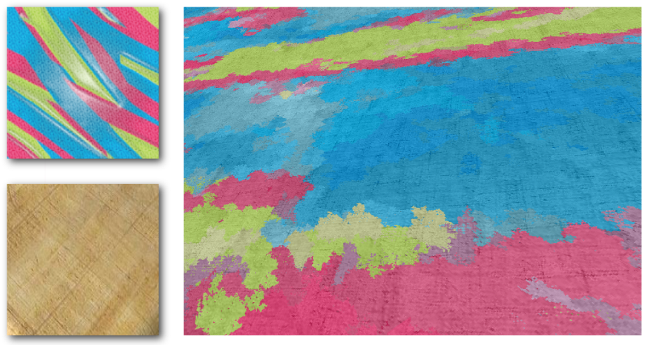


Figure 13: *Failure of the fragmentation because of too low contrasts. Seams are visible after color transfer.*

9 Conclusions

We have presented a framework for modeling example-based infinite textures generated on-the-fly. It improves over tiling by predefining several exchangeable contents that can be randomly selected during real-time texturing. It thus breaks repetition effects while remaining compact. Multi-scale is also smartly integrated in the process. Indeed, we define a feature-aware color transfer method that produces smooth and consistent transitions between multiple scales. It can moreover be enhanced with relief. The overall coherence relies on the detection of small features (fragments) that i) have salient borders used to hide seams for content exchange or color transfer, ii) encompass a single unitary feature used for consistent color transfer, iii) are clustered for defining patches with exchangeable contents encompassing several features, thus breaking repetition. Results report unprecedented quality and speed for infinite texturing using such a compact example-based representation. We also provide a simple yet efficient tool for interactively creating an elevation map without expert knowledge requirements.

This paper calls for many future works. Firstly, our segmentation algorithm fulfills our requirements for this work, but can probably be improved. It would be interesting to take advantage of some good properties of texture segmentation algorithms which are not trivial to adapt to our purpose. Moreover, our color space defines a few dominant colors and separates fine variations. We have shown that it provides flexibility in color transfer and blending. It would be worth to go deeper into the modeling of transfers: an interactive transfer modeling tool would be useful.

So far, we have considered only one single texture per scale. A valuable extension would consist in dealing with multiple textures for each scale. Imagine a coarse scale being a brick wall: such an approach could differentiate, at the fine scale, one texture for the bricks and another for the concrete.

Relief reconstruction using local descriptors also opens the way to some future directions. We believe that similar descriptors may be used to reconstruct a geometric information not limited to height fields. We could for instance imagine that the user models a coarse polygonal shape of the texture element and the system reconstructs distributions and deformations from the image, so as to synthesize similar textures in full 3D.

Finally, we think that paying more attention to perception could help to further improve texture synthesis for both quality and speed.

Acknowledgements

We thank Olivier Génevaux for his technical support, as well as the reviewers and referees for their precious feedback.

References

ACHANTA, R., SHAJI, A., SMITH, K., LUCCHI, A., FUA, P., AND SUSSTRUNK, S. 2012. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Trans. Pattern Anal. Mach. Intell. 34*, 11 (Nov.), 2274–2282.

BOSCH, C., LAFFONT, P.-Y., RUSHMEIER, H., DORSEY, J., AND DRETTAKIS, G. 2011. Image-guided weathering: A new approach applied to flow phenomena. *ACM Trans. Graph. 30*, 3 (May), 20:1–20:13.

BOURQUE, E., AND DUDEK, G. 2004. Procedural texture matching and transformation. *Comp. Graph. Forum 23*, 3, 461–468.

BROOKS, S., AND DODGSON, N. 2002. Self-similarity based texture editing. In *ACM SIGGRAPH 2002 papers*, ACM, New York, NY, USA, 653–656.

CHEN, Y., TONG, X., WANG, J., LIN, S., GUO, B., AND SHUM, H.-Y. 2004. Shell texture functions. In *ACM SIGGRAPH 2004 Papers*, ACM, New York, NY, USA, 343–353.

COHEN, M. F., SHADE, J., HILLER, S., AND DEUSSEN, O. 2003. Wang tiles for image and texture generation. In *ACM SIGGRAPH 2003 Papers*, ACM, New York, NY, USA, 287–294.

DISCHLER, J.-M., MARITAUD, K., AND GHAZANFARPOUR, D. 2002. Coherent bump map recovery from a single texture image. In *Proc. of Graphics Interface*, 201–208.

EBERT, D. S., MUSGRAVE, F. K., PEACHEY, D., PERLIN, K., AND WORLEY, S. 2002. *Texturing and Modeling: A Procedural Approach*, 3rd ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

GALERNE, B., LAGAE, A., LEFEBVRE, S., AND DRETTAKIS, G. 2012. Gabor noise by example. *ACM Trans. Graph. 31*, 4 (July), 73:1–73:9.

GERVAUTZ, M., AND PURGATHOFER, W. 1990. A simple method for color quantization: Octree quantization. In *Graphics Gems*, A. S. Glassner, Ed. Academic Press, 287–293.

GILET, G., AND DISCHLER, J.-M. 2010. Procedural descriptions of anisotropic noisy textures by example. In *Eurographics 2010, Short Paper*, Eurographics Association.

HAN, C., RISSER, E., RAMAMOORTHI, R., AND GRINSPUN, E. 2008. Multiscale texture synthesis. In *ACM SIGGRAPH 2008 papers*, ACM, New York, NY, USA, 51:1–51:8.

KOPF, J., COHEN-OR, D., DEUSSEN, O., AND LISCHINSKI, D. 2006. Recursive wang tiles for real-time blue noise. In *ACM SIGGRAPH 2006 Papers*, ACM, New York, NY, USA, 509–518.

KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: image and video synthesis using graph cuts. In *ACM SIGGRAPH 2003 Papers*, ACM, New York, NY, USA, 277–286.

LAGAE, A., AND DUTRÉ, P. 2005. A procedural object distribution function. *ACM Trans. Graph. 24*, 4 (Oct.), 1442–1461.

LAGAE, A., AND DUTRÉ, P. 2006. An alternative for Wang tiles: Colored edges versus colored corners. *ACM Trans. Graph. 25*, 4 (Oct.), 1442–1459.

LAINE, S., AND KARRAS, T. 2010. Efficient sparse voxel octrees. In *Proc. of the 2010 ACM SIGGRAPH symp. on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, 55–63.

LEFEBVRE, S., AND HOPPE, H. 2005. Parallel controllable texture synthesis. In *ACM SIGGRAPH 2005 Papers*, ACM, New York, NY, USA, 777–786.

LOSASSO, F., AND HOPPE, H. 2004. Geometry clipmaps: terrain rendering using nested regular grids. In *ACM SIGGRAPH 2004 Papers*, ACM, New York, NY, USA, 769–776.

LOWE, D. G. 2004. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision 60*, 2 (Nov.), 91–110.

MA, W.-C., JONES, A., CHIANG, J.-Y., HAWKINS, T., FREDERIKSEN, S., PEERS, P., VUKOVIC, M., OUHYOUNG, M., AND DEBEVEC, P. 2008. Facial performance synthesis using deformation-driven polynomial displacement maps. In *ACM SIGGRAPH Asia 2008 papers*, ACM, New York, NY, USA, 121:1–121:10.

MÜLLER, P., ZENG, G., WONKA, P., AND VAN GOOL, L. 2007. Image-based procedural modeling of facades. In *ACM SIGGRAPH 2007 papers*, ACM, New York, NY, USA, 85:1–85:9.

OMER, I., AND WERMAN, M. 2004. Color lines: image specific color representation. In *Proc. of the 2004 IEEE conf. on Computer Vision and Pattern Recognition*, IEEE Computer Society, Washington, DC, USA, 946–953.

PERAZZI, F., KRÄHENBÜHL, P., PRITCH, Y., AND HORNUNG, A. 2012. Saliency filters: Contrast based filtering for salient region detection. In *Proc. of the 2012 IEEE conf. on Computer Vision and Pattern Recognition*, IEEE Computer Society, Washington, DC, USA, 733–740.

PERLIN, K. 1985. An image synthesizer. In *ACM SIGGRAPH 1985 papers*, ACM, New York, NY, USA, 287–296.

POLICARPO, F., OLIVEIRA, M. M., AND COMBA, J. A. L. D. 2005. Real-time relief mapping on arbitrary polygonal surfaces. In *ACM SIGGRAPH 2005 Papers*, ACM, New York, NY, USA, 935–935.

TOU, J. Y., TAY, Y. H., AND LAU, P. Y. 2009. Recent trends in texture classification: a review. In *Proc. of the 2009 symp. on Progress in Information and Communication Technology*.

TREIB, M., REICHL, F., AUER, S., AND WESTERMANN, R. 2012. Interactive editing of gigasample terrain fields. *Comp. Graph. Forum 31*, 2pt2 (May), 383–392.

WEI, L.-Y., LEFEBVRE, S., KWATRA, V., AND TURK, G. 2009. State of the art in example-based texture synthesis. In *Eurographics 2009, State of the Art Report*, Eurographics Association.

WU, H., DORSEY, J., AND RUSHMEIER, H. 2011. Physically-based interactive bi-scale material design. In *ACM SIGGRAPH Asia 2011 papers*, ACM, New York, NY, USA, 145:1–145:10.

ZENG, G., MATSUSHITA, Y., QUAN, L., AND SHUM, H.-Y. 2005. Interactive shape from shading. In *Proc. of the 2005 IEEE conf. on Computer Vision and Pattern Recognition*, IEEE Computer Society, Washington, DC, USA, 343–350.

ZHANG, R., TSAI, P.-S., CRYER, J. E., AND SHAH, M. 1999. Shape from shading: A survey. *IEEE Trans. Pattern Anal. Mach. Intell. 21*, 8 (Aug.), 690–706.